A GRAPH THEORETIC APPROACH TO THE CLASS SCHEDULING PROBLEM

Charles Lewis Deitrick

United States
Naval Postgraduate School

THESIS

A GRAPH THEORETIC APPROACH
TO
THE CLASS SCHEDULING PROBLEM

by

Charles Lewis Deitrick

Thesis Advisor: U. R. Kodres

June 1971

Approved for public release; distribution unlimited.

A Graph Theoretic Approach to The Class Scheduling Problem

by

Charles Lewis Deitrick
Lieutenant, United States Navy
B.S., Pennsylvania State University, 1965

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1971

ABSTRACT

Two algorithms for coloring large order graphs by partitioning, as related to class scheduling with a computer, are developed. Although, the two main algorithms failed to produce acceptable results for application to class scheduling, a coloring algorithm developed for use in the two main algorithms, is an improvement over known existing coloring algorithms.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF TABLES

LIST OF DRAWINGS

Figure

I. INTRODUCTION

The procedure of scheduling classes at the Naval Postgraduate School, Monterey, California, differs from most academic institutions. At most schools a preassigned schedule of courses is presented to the students, and the students schedule themselves to courses which do not conflict. At the Naval Postgraduate School the student is required to take predetermined courses. This presents difficulties in developing a class schedule in which neither student nor instructor will have a class conflict. It is this problem that is investigated.

Manual production of a class schedule without conflicts is difficult and time consuming. This method consists of successively entering courses into the schedule. If an entry conflicts with previous entries, then the previous entries are modified to make the new entry possible. This process is continued until all courses are entered into the schedule. The time and labor involved in manual methods has created interest in the use of the speed of the general purpose digital computer in developing class schedules.

Many methods of applying the computer to class schedule production have been proposed or attempted. Some methods produce satisfactory results in restricted cases. These methods stress either preknowledge of the existence of a solution or obtain an acceptable solution for as many courses as possible within a reasonable computer time.

Appleby, Blake, and Newman [1] attempted to produce class schedules with the computer. Their proposed techniques include:

(1) a random trial solution updated to eliminate conflicts,

(2) a trial of all combinations until a satisfactory schedule is achieved,

(3) a random buildup of entries until a conflict occurs, then modification of previous entries until the conflict is resolved,

(4) a heuristic approach.

They had modest success especially in high school schedules. Difficulties arose in knowing when an entry made the completion of class schedule impossible.

A method using a 3-dimensional scheduling array with elements of zeroes or ones is described by Csima and Gottlieb [3]. The method is inefficient due to much computer time being used in setting up and manipulating the matrices. Although a proof for the existence of a solution in general is lacking, there are theorems which guarantee solutions for special cases.

A method using graph theory is described by Mack [4]. The method relates course or class conflicts to connections within an abstract graph and attempts a solution using graph coloring techniques. The method was applied to data from the second quarter of the 1967-1968 academic year at the Naval Postgraduate School with no acceptable results obtained.

Mack's method used the Welsh-Powell [6] node coloring algorithm applied to the coloring problem.

In treating the scheduling problem as a graph theory problem, classes are represented as nodes of an abstract graph, and class conflicts are represented by a connection between nodes. When the scheduling problem has been transformed to an abstract graph, the original problem may then be treated as a node coloring problem in graph theory, where colors are assigned to all nodes of the graph such that no two nodes, which are connected, are assigned the same color. When an acceptable node coloring solution is attained, the various hours of the weekly schedule may be assigned to colors and the weekly schedule for the classes is achieved. This requires that an acceptable node coloring be found.

Not all solutions to the node coloring problem are acceptable as a solution to the scheduling problem. One solution to the node coloring problem is to assign a distinct color to each node, but this would be totally unacceptable as a solution to the scheduling problem. Thus, a minimum or near minimum set of colors must be found. This set of colors is not easily found, since the number of possible solutions to the node coloring problem is combinatorially related to the number of nodes in the abstract graph. Therefore, if the number of nodes is small, the solution is easier to find.

This paper will treat the scheduling problem as a graph theory problem. The approach is to attempt a solution

of the problem by breaking up the graph into several small subgraphs, then find solutions to the small subgraphs and combine them to form a solution to the original graph.

The scheduling problem at the Naval Postgraduate School has inherent restrictions due to the nature of the school. Most students have preassigned curriculums, thus courses are taken in a prescribed sequence. In addition, the students have a prescribed length of time to accomplish their studies. This, along with restrictions due to instructor requirements, introduces many constraints into the scheduling problem.

The constraints of preassigned curriculums may be used to an advantage in solving the problem. Most students assigned to the Naval Postgraduate School are assigned for the pursuit of a particular degree, and most of the courses for a particular degree are associated with a department. Thus, most conflicts and restrictions would be departmental, with relatively few interdepartmental conflicts and restrictions.

The method proposed in Section III for solving the scheduling problem at the Naval Postgraduate School attempts to solve the scheduling problem for individual departments using departmental restrictions. The interdepartmental restrictions are then used to combine the departmental solutions into a solution to the total scheduling problem.

In Section II of this paper, the scheduling problem is developed in graph theoretic terms. The basic concepts and

definitions, used in formulating and solving the corresponding graph theory problem, are presented.

In Section III, the principal algorithms and the results obtained, are presented. Section IV gives the conclusions. The Appendix contains the PL/1 listing of algorithms used.

II. DEFINITION OF THE PROBLEM

A. PRELIMINARY DEFINITIONS

In order to clarify the problem, as stated in graph theoretic terms, the following definitions taken from Busacker and Saaty [2] and Ore [5], are needed.

Definition 1.

An unordered product of a set S with itself, denoted by (S&S), is defined as the set of all unordered pairs (s&t) where $s \varepsilon S$, $t \varepsilon S$ and (s&t) = (t&s).

Definition 2.

An abstract graph is defined as a non-empty set V, a (possibly empty) set E, and mapping ϕ of E into (V&V). The elements of V and E are called vertices or nodes and edges or arcs of the graph respectively, and ϕ is called the incidence mapping associated with the graph. ϕ is a mapping from the set of edges to the set of unordered pairs of vertices (V&V), thus, $\phi(e) = (v\&w)$, where $v, w \varepsilon V$. This convention allows graphs to contain multiple edges or parallel arcs, i.e., $\phi(e_1) = \phi(e_2) = (v\&w)$, where $e_1, e_2 \varepsilon E$. A graph will usually be denoted by G or (V,E).

For the purpose of this paper, multiple edges or parallel arcs are considered as one edge. Thus, by specifying the endpoints of an edge, the edge itself may be specified. Therefore, the abbreviated notation, e = (v&w), may be used for, $\phi(e) = (v\&w)$.

Definition 3.

Vertices v and w are said to be adjacent vertices if $\phi(e) = (v\&w)$ for at least one edge e.

Definition 4.

An edge e is said to be incident with v and w if $\phi(e) = (v\&w)$ for some v and w. This relation is denoted by $e = (v\&w)$ and read e joins v and w.

Definition 5.

The number of edges incident with a vertex v is called the degree of v and denoted $\delta(v)$. A vertex is said to be isolated if $\delta(v) = 0$.

Definition 6.

The number of vertices in the set V, is defined as the order of the graph G, denoted by $o(G)$.

Definition 7.

A complete graph is defined as a graph where each node is adjacent to every other node of the graph.

Definition 8.

A graph $G' = (N',E')$ is defined as a subgraph of $G = (N,E)$ if all nodes of N' are contained in N, and all edges in E' are contained in E, and furthermore, for $e' \in E'$ $\phi(e') = (v\&w)$, where $v, w \in N'$.

Definition 9.

A course conflict is said to exist between two courses if they cannot be scheduled at the same time.

B. THE SCHEDULING PROBLEM

The scheduling problem at the Naval Postgraduate School consists of taking a set of students S, a set of instructors P, a set of courses C, and deriving a weekly schedule so that each course is assigned to an appropriate number of time periods within the five 9-hour days. This must be done so that no student or instructor is assigned to two courses during the same time period of a day. This problem is approached by formulating and solving a parallel problem in graph theory.

Definition 10.

A section is a group of students who have the same schedule. For definiteness, a section is represented by a student, belonging to that section.

An abstract graph G = (N,E) represents conflicts between courses introduced by the students $S = \{s_1, s_2, s_3, \ldots, s_n\}$, and the set of instructors $P = \{p_1, p_2, p_3, \ldots, p_m\}$. Let $N = \{c_1, c_2, c_3, \ldots, c_l\}$ be the set of courses.

(1) If student s_1 is required to take courses $c_1, c_2, \ldots, c_k$ then G contains a complete graph on these nodes as a subgraph, i.e., $e_1 = (c_1 \& c_2)$, $e_2 = (c_1 \& c_3), \ldots, e_{k-1} = (c_1 \& c_k)$, $e_k = (c_2 \& c_3), \ldots, e_{\frac{k(k-1)}{2}} = (c_{k-1} \& c_k)$, belong to E.

(2) If instructor p_i is assigned to instruct courses $c_1, c_2, c_3, \ldots, c_k$, then G contains a complete graph on these nodes as a subgraph, same as in paragraph (1) above.

Definition 11.

The above abstract graph G = (N,E) is defined as the conflict graph.

The following is an example of a conflict graph for the data given:

(1) Student s_1 is required to take courses c_1, c_2, and c_3.

(2) Student s_2 is required to take courses c_2, c_3, and c_4.

(3) Instructor p_1 is assigned to instruct courses c_1 and c_3.

(4) Instructor p_2 is assigned to instruct courses c_2 and c_4.

As defined above, $G = (N,E)$, with $N = \{c_1, c_2, c_3, c_4\}$, and $E = \{e_1 = (c_1 \& c_2), e_2 = (c_1 \& c_3), e_3 = (c_2 \& c_3), e_4 = (c_2 \& c_4), e_5 = (c_3 \& c_4)\}$, is the conflict graph. Since the conflict $(c_2 \& c_3)$ is entered for s_1, it is not repeated for s_2, only new conflicts are added.

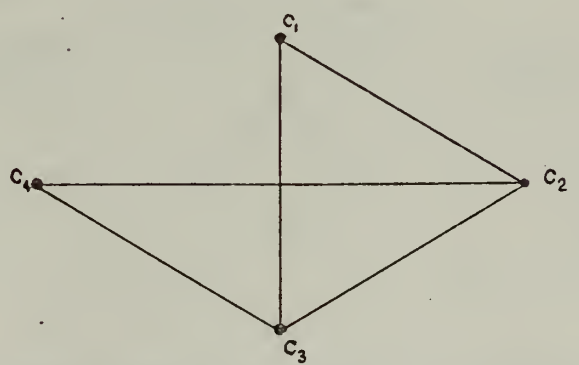


Figure 1. A Course Conflict Graph.

The course conflict graph, as exemplified above, must be expanded to achieve a solution to the scheduling problem. A course may have a number of recitation periods which must be scheduled on different days, or a lab period which consists of one or more hours on the same day, or it may have both. Thus, the course conflict graph must be expanded so

it contains a node to represent each hour period required by a course. A set of nodes, $N = \{n_i \mid i=1 \text{ to } k\}$, where k is the total number of hours required, is used to represent each contact hour of a course. The resulting graph is called an hourly conflict graph.

Figure 2 illustrates the hourly conflict graph derived from the previous example, and the following data:

(1) Course c_1 is a 2 hour lab,

(2) Course c_2 is 3 hours of recitation,

(3) Course c_3 is 2 hours of recitation and 2 hours of lab,

(4) Course c_4 is 1 hour of recitation.

Assign nodes in G as follows: $c_1 \simeq \{n_1, n_2\}$, $c_2 \simeq \{n_3, n_4, n_5\}$, $c_3 \simeq \{n_6, n_7, n_8, n_9\}$, $c_4 \simeq \{n_{10}\}$.

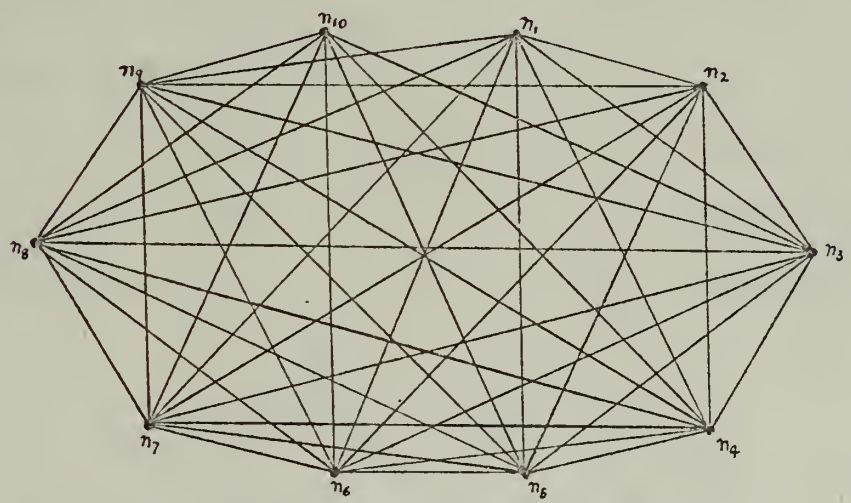


Figure 2. An Hourly Conflict Graph.

The scheduling of courses is not the only restraint on the scheduling problem. If a student is an aviator, he must have a consecutive sequence of five hours scheduled for flying.

In addition, instructors may be scheduled for research time, special lectures, and departmental meetings. These additional restrictions can be built into the schedule by considering them as additional courses.

The constraints imposed upon the scheduling problem can be categorized into three groups, namely, daily conflicts, hourly conflicts, and consecutive hours. Daily conflicts result from a course having multiple recitation periods. Each recitation period must be scheduled on a different day. All other conflicts may be considered as hourly conflicts. Consecutive hours are hourly conflicts with the additional constraints that the hours be consecutive and on the same day. Figure 3 is an example of a conflict graph with two 3-hour courses (c_1,c_2) and a 2-hour lab (c_3). The nodes corresponding to c_1 conflict with c_2, and c_2 conflicts with c_3.

$c_1 \simeq \{n_1, n_2, n_3\}$

$c_2 \simeq \{n_4, n_5, n_6\}$

$c_3 \simeq \{n_7, n_8\}$

_____ hourly conflict

___x___ daily conflict

N_i consecutive hours

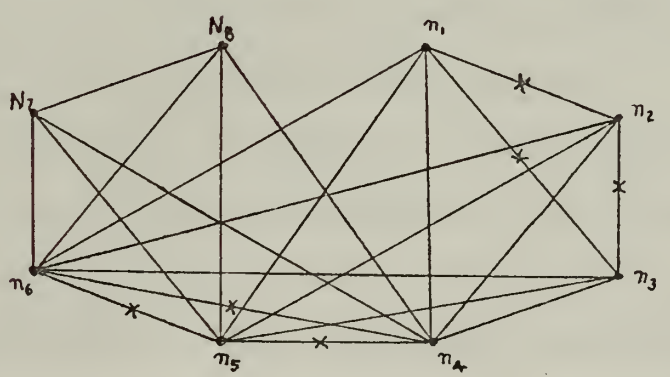


Figure 3. A Conflict Graph Illustrating Hourly Conflicts, Daily Conflicts, and Consecutive Hours.

Definition 12.

A graph is said to be K-colorable when there exists a partitioning of its nodes, N, into classes C_1, C_2, C_3,...,C_k such that $\bigcup_i C_i = N$, no node is a member of more than one class, and where two nodes of the same class are not connected.

Definition 13.

The minimum number K for which a graph is K-colorable is defined as the chromatic number of the graph.

The method of solution to the scheduling problem is to color the graph theory model with K colors. A graph with chromatic number K, may have K colors assigned to its nodes so that no two adjacent nodes have the same color. All nodes of the same class C_i, as described in Definition 13, will have the i^{th} color assigned to them. A coloring of the graph theory model of the scheduling problem may be used to solve the scheduling problem. Course conflicts are represented in the graph theory model by an edge in the graph. Thus, if time periods of the schedule are assigned to the K colors of the graph, courses that conflict with one another will be scheduled for different time periods.

In graph coloring, the colors are usually sequentially numbered starting at 1 to some number K. This method of numbering the colors is not directly applicable for use in the scheduling problem. The scheduling problem has the constraints previously referred to as daily conflicts, hourly conflicts and consecutive hours that must be satisfied. In

addition to these constraints, the restrictions of the schedule which require that all classes be scheduled within 9 hour periods per day in a 5 day week, give rise to a modified color notation. A method of color notation which is suitable to the scheduling problem consists of color vectors. In this application, a set of two digit numbers may be used. Thus, by letting the tens digit represent the day and the units digit represent the hour of the day, the time periods of the schedule may be represented. The set of numbers 11 thru 59 may be used, with the numbers 20, 30, 40, and 50 being invalid. In this notation the number 26 would represent the sixth hour on the second day. The invalid numbers would signify the bounds which separate the days. For example, the numbers 19, 21, and 22 could not be used for three consecutive hours, since the numbers are not sequential. By using these color vectors, a check on the observance of the constraints may be made easily by checking the color vector. In addition, these color vectors may be used in producing the desired schedule.

There usually are many solutions to the node coloring problem, even when the number of colors is equal to the chromatic number K. Exhaustively generating all possible solutions to the node coloring problem cannot be used except in very small graphs. There is presently no known method which guarantees a coloring with the chromatic number for a graph with 1,000 to 2,000 nodes.

The graph that results from the scheduling problem is of large order, that is from 1,000 to 2,000 nodes. Only a

small subset of the solutions to the node coloring problem would be acceptable as solutions to the scheduling problem. These solutions are likely to have a minimum or near minimum number of colors. Thus, an algorithm for coloring the nodes with a minimum or near minimum number of colors without looking at all solutions would be helpful in solving the scheduling problem.

Definition 14.

The connectivity between two sets of nodes is defined as the percentage of the maximum possible number of edges connecting the two sets of nodes.

Definition 15.

The maximum number of elements in E, for a graph G = (N,E) with n elements in N, is $\frac{n(n-1)}{2}$.

Definition 16.

Internal connectivity is defined as the connectivity of a node in a set to other nodes of the same set.

Definition 17.

External connectivity is defined as the connectivity of a set of nodes of a graph to the nodes of its complement.

Due to the nature of the Naval Postgraduate School and the special restrictions within the schedule, the graph which is produced from the schedule has special characteristics. The nodes representing class hours can be partitioned into subsets. These subsets can be described as having relatively low external connectivity.

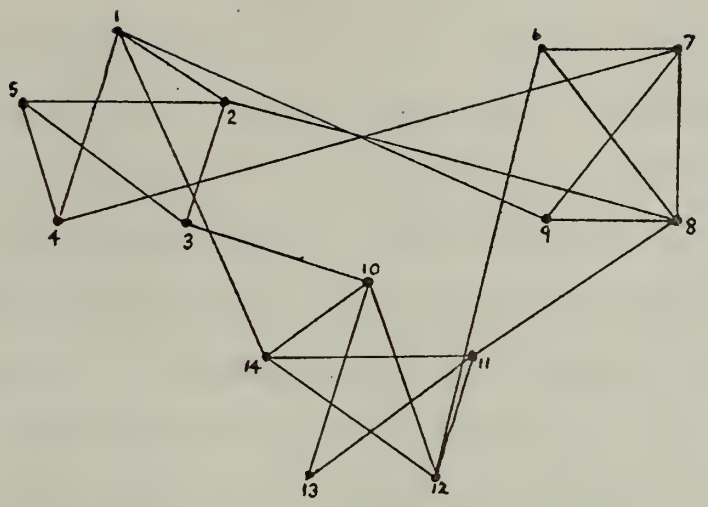


Figure 4. A Graph that may be Partitioned into Subsets.

Figure 4 is an example of a graph that may be partitioned into subsets. There are 14 nodes, thus the maximum number of edges would be 14(14-1)/2, or 91. The set of nodes N can be partitioned as follows:

$$N = N_1' \cup N_2' \cup N_3',$$

where

$N_1' = \{1, 2, 3, 4, 5\}$,

$N_2' = \{6, 7, 8, 9\}$,

$N_3' = \{10, 11, 12, 13, 14\}$.

N_1' contains 5 nodes, thus it could contain a maximum of 10 edges connecting its nodes. It only contains 6, thus internal connectivity is 60 percent. Each of the 5 nodes of N_1' can connect to each of the remaining 9 nodes for a maximum of 45 external edges. Only 5 external edges exist, thus the external connectivity N_1' is 11 percent. The partitions N_2' and N_3' can be described in the same manner.

When a graph lends itself to partitioning of the type where the external connectivity is very low, the difficulties encountered in coloring large order graphs may be reduced by first coloring the partitions independently. The coloring of the whole graph is then achieved by using the external connections of the partitions to combine the partition colorings. The difficulty of finding a minimal or acceptable solution still exists, because coloring and combining the partitions still requires a minimal or a near minimal solution.

III. EXPERIMENTAL PROCEDURES

A. THE ALGORITHMS

The algorithm for coloring the abstract graph by partitions requires a coloring algorithm which is slightly different from the existing coloring algorithms of which the Welsh-Powell algorithm [6] is a well known example. In addition, two techniques of combining the partition solutions were examined. Thus, the development of the overall algorithm is presented in stages with the evaluation at each stage.

1. The Coloring Algorithm

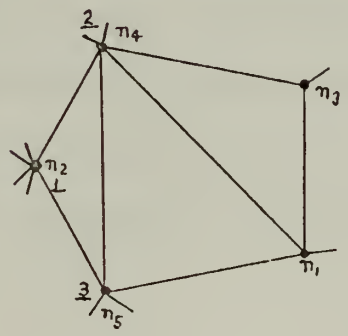
The algorithm developed for this application is designed to handle certain conditions which occur in graph coloring. Under these conditions, referred to as forward color conflicts, the assignment of a particular color to a node, will connect all present colors of the graph to an uncolored node. The colors connected to this uncolored node are referred to as forward conflict colors. Thus, when this uncolored node is colored, an increase in colors is required. In some situations, the forward color conflict may be avoided by using the details of the graph in selecting colors.

Forward color conflicts may occur within a partition or may result from inter-partition connection. An example of a forward color conflict is illustrated in Figure 5. Consider Figure 5 as a sub-section of a graph to be

colored. The nodes of the entire graph are ordered such that n_2, n_4, and n_5 are assigned colors 1, 2, and 3 respectively, with n_3 as the next node to be assigned a color. n_3 may be assigned either color 1 or 3 but not color 2, since it is adjacent to a node which has previously been assigned color 2. Two situations occur from the choice of color for n_3.

(1) If color 1 is assigned, then n_1 will be adjacent to colors 1, 2, 3, and an additional color 4 would be required.

(2) If color 3 is assigned, then n_1 will be adjacent to colors 2 and 3, and color 1 may be assigned to it, thus avoiding the increase in the number of colors as in Option (1).



⊥color assigned

Figure 5. Illustration of Forward Color Conflict.

The coloring algorithm developed for this application, known as "Look-ahead" algorithm, has a capability to resolve the forward color conflict by using one level of look-ahead when a color assignment option arises. The algorithm assigns colors to the nodes by assigning color 1 to the node of highest degree and then sequentially assigning

a color to each node in the list of nodes. The color for each node is selected from the list of previously assigned colors, such that the color is different from the colors of the adjacent nodes. If no color is available, a new color is added to the list of colors. In the situation where a forward color conflict occurs, the algorithm looks one level ahead. The available color, with the maximum number of nodes contributing to a forward conflict color, is assigned.

An example of this look-ahead principle is shown in Figure 6. Assume that colors 1 thru 3 have been assigned to a graph in Figure 6, and that node n_1 is being considered for a color. Colors 2 and 3 are available. Color 1 is a first level conflict color, resulting from node n_9. Color 2 is a forward conflict color, resulting from node n_7, and color 3 is a forward conflict color resulting from nodes n_4 and n_6. Thus, color 3 is the color with the maximum number of nodes contributing to a forward conflict color, hence it is assigned to n_1.

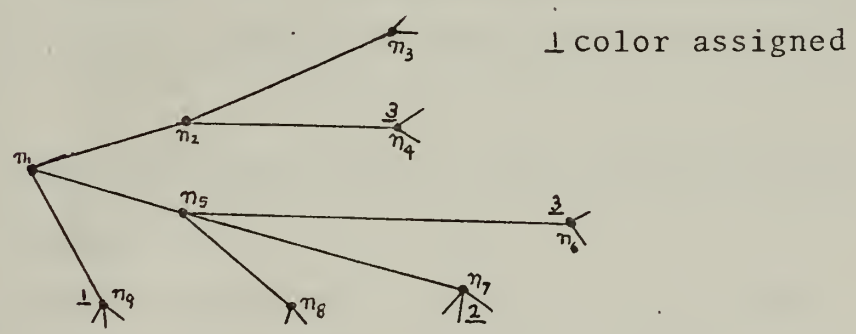


Figure 6. Illustration of Forward Conflicts.

The Look-ahead algorithm is now described as it pertains to a general abstract graph.

Let G = (N,E) be an arbitrary abstract graph with $N = \{n_1, n_2, n_3, \ldots, n_o\}$ as the set of nodes and $E = \{e_1, e_2, e_3, \ldots, e_m\}$ as the set of edges, where $e_i = (n_j \& n_k)$. Let the degree of node n_i be d_i, and assume that the nodes have been numbered so that $d_1 \geq d_2 \geq d_3 \geq, .., \geq d_n$. Let $C = \{c_1, c_2, c_3, \ldots, c_k\}$ be the set of colors. Then the Look-ahead algorithm may be described by the following steps.

(1) n_1 is assigned color c_1, k is set to 1. Initialize i=2, j=1, $C = \{c_1\}$.

(2) Find the set of colors A, not available for assignment to the next uncolored node n_i. $c_j \in A$ for all nodes n_d such that n_d is adjacent to n_i, and n_d has been assigned color c_j, $(j \leq k)$.

(3) If the set of available colors, (C-A), is empty, n_i is assigned a new color c_{k+1}, k is incremented by one, and step (6) is taken. Otherwise go to step (4).

(4) If the set (C-A) contains only one element c_j, assign color c_j to n_i and go to step (6), else go to step (5).

(5) (The look-ahead step). Find the set of forward conflict colors.

Let n_i be the node which we wish to color next. If there is a sequency of two edges in E, $(n_i \& n_d)$, $(n_d \& n_p)$, which connects n_i to a colored node n_p via an uncolored node

n_d, then the color c_j of n_p is called a conflict color, and n_p is called a conflict node.

There exists a color c_m with the maximal number of conflict nodes. If there are several colors with the same maximal number of conflict nodes, then the color with a lower index is chosen to color node n_i. Continue with step (6).

(6) If node n_i is not the last node, then update i and return to (2). Otherwise all nodes have been colored.

2. Combination by Matching

In coloring a graph by partitions, an abstract graph G = (N,E) of the form shown in Figure 4, which may be partitioned into m partitions, is assumed. The characteristics of the graph are such that the ratio of the partitions' internal connectivity to external connectivity is high. Thus the set of nodes N is partitioned into classes P_i for i=1 to m, with $\bigcup_i P_i = N$, and there are relatively few edges connecting nodes in P_i to nodes in P_j for $P_i \neq P_j$. Each partition P_i is then colored using the Look-ahead coloring algorithm. Each partition P_j, is considered as a separate graph except when external connections are used to resolve forward color conflicts.

Assume that each partition P_i is colored with K_i colors, and also that the partitions are numbered so that $K_1 \geq K_2 \geq K_3 \geq \ldots \geq K_m$. Thus, combining the partitions in decreasing order will combine the partitions with the highest number of colors first. The partition color solutions

are then successively matched to the already combined partitions to form a coloring solution for the total graph.

Definition 18.

The complement of a graph G = (N,E) on n nodes, denoted by $\overline{G}$, is obtained by deleting from a complete graph on n nodes, those edges that occur in the original graph G.

Definition 19.

Given a graph G = (N,E) on n nodes, in which the nodes have been partitioned into non-empty classes S_i for i=1 to k, and $\bigcup_i^k S_i = N$. A new graph G' = (S',F) may be constructed, where $n_i' \varepsilon S'$, for i=1 to k, and where each n_i represents a class of nodes S_i from the original graph G. $f \varepsilon F$ if there exist an edge $e \varepsilon E$, where $e = (n_s \& n_t)$, $n_s \varepsilon S_u$, $n_t \varepsilon S_v$ for $u \neq v$. Graph G' is defined as a coalesced graph.

The solution to the coloring problem is constructed by successively matching the colors of the partition solutions, with the previously assigned final colors. A graph G' is constructed from the original graph G. All nodes of the original graph, with a final color assigned, are coalesced and represented by a complete subgraph on K nodes, where K is the number of final colors. The nodes of the partition to be matched P_i, are coalesced and represented by a complete subgraph on K_i nodes. Thus, the final colors and the partition colors are represented by the nodes of the two complete subgraphs. Edges are then constructed between the nodes representing final colors and the nodes representing partition P_i's colors. If a node in partition P_i,

with partition color c_j, is connected to a node with a final color, then an edge is constructed between the nodes representing the partition color c_j and the final color. These edges represent color conflicts, when one is assigning a final color to the nodes with a particular partition color.

Definition 20.

A graph G = (N,E) is said to be bipartite, if its nodes N can be partitioned into two disjoint sets N_1 and N_2, such that every edge e, $e \varepsilon E$, has one endpoint in N_1 and the other in N_2.

Definition 21.

Nodes n_i and n_j of a graph G = (N,E) may be matched if E contains an edge $e = (n_i \& n_j)$, $i \neq j$.

Definition 22.

A sequence of edges $e_1, e_2, e_3, \ldots, e_m$ on a sequence of nodes $n_1, n_2, n_3, \ldots, n_{m+1}$, such that $e_i = (n_i \& n_{i+1})$, is defined as an edge chain.

The graph $\overline{G}'$ is a bipartite graph with the K nodes, representing the final colors, as one set of nodes, and the K_i nodes, representing the partition colors, as the other set of nodes. The selection of final colors for nodes of a partition with partition color c_j, for j=1 to K_i, is selected by maximal matching of the nodes within the bipartite graph $\overline{G}'$.

Maximal matching is achieved by selecting an edge in $\overline{G}'$ and assuming it as a solution. Then the existence of an alternating edge chain, an edge chain in which alternate edges are in the present solution, beginning at an

unmatched node and ending on another unmatched node, is proof that the matching is not maximal. A better solution for maximal matching is to delete from the present solution those edges which are part of the alternating edge chain and add the edges which are not part of the present solution. The process of testing for an alternating edge chain is then repeated. If an alternating edge chain cannot be found maximal matching is achieved. This procedure is described in more detail in Ore [5], pp. 132-137.

The partition solutions are combined by assuming the partition colors of P_1 are final colors, then a matching of nodes with partition P_2 is began. If a node, in the bipartite graph representing a partition color c_j, is not matched with a node representing a final color, a new final color is added. Then all nodes of the partition with partition color c_j are assigned the new final color. When all nodes of the partition are assigned final colors, the next partition is processed. When final colors have been assigned to nodes in all partitions, a coloring of the original graph has been constructed.

3. Combination by Coloring Coalesced Graph

In coloring a graph by partitions using the coalesced graph coloring for combination, a graph G = (N,E) of the form used in combination by matching, is assumed. It is further assumed that the partitions P_i have been colored with K_i, for i=1 to m. The ordering on the number of colors in the partition is not required by this method.

A graph G" = (N",E"), a coalesced graph of graph G, is constructed using the partition color solutions K_1, K_2, K_3,....K_m. The set of nodes N" contains a node to represent each color of all partitions. Thus the order of G" is $\sigma = K_1 + K_2 + K_3 + \ldots + K_m$, with the set of nodes n_i for i=1 to K_1 representing colors of partition P_1, n_i for $i=K_1+1$ to K_1+K_2 representing colors partition P_2, n_i for $i=K_1+K_2+1$ to $K_1+K_2+K_3$ representing colors of partition P_3, etc. The set of edges E" is constructed by considering the edges of the original graph G which connect nodes in different color sets. Thus, if $e \in E$ and connects a node of one color to a node of a different color, then an edge e" is added to E". The edge e" connects the nodes in N" representing the two colors. In addition an edge is added to E" for all edges external to the partitions, since an edge connecting nodes of different partitions is considered as connecting different colors. Since an optimal or near optimal coloring of each partition is assumed, the color sets within each partition mutually connect to each other. Thus the set of nodes representing the colors of a partition comprise a complete subgraph of G".

The coalesced graph G" is then colored by ordering the nodes of N" in descending order by degree of the nodes, and submitting it to the Look-ahead coloring algorithm. The number of colors required to color G" will be the number of colors required for coloring the nodes of G. The nodes of G" which have a common color, and which represent nodes of

G which may be colored with the same color, are all combined into a common class and a single color assigned.

B. RESULTS

The above algorithms were programmed in PL/1, as internal subroutines, along with supporting subroutines. These subroutines were then called in various sequences to evaluate the effectiveness of the algorithms. The PL/1 listings for the three algorithms are included in this paper.

The algorithms were evaluated using randomly generated graphs. These graphs were generated by specifying the following parameters:

(1) Order of the graph,

(2) Number of partitions in the graph,

(3) Percentage of internal connections for a partition,

(4) Percentage of external connections for a partition.

The nodes are randomly assigned to a partition and then all node pairs are considered for an edge in the graph. A random number generator is used along with the percentage of internal connections or percentage of external connections, depending upon whether the two nodes are in the same partition or not. A random graph, non-partitioned, may be generated by specifying only one partition.

The graph is contained in a matrix, named CONGRAPH, which is common to all subroutines. Each row in the matrix represents a node of the graph. The edges of the graph are represented by listing the connected nodes sequentially in the row. The nodes of a partition are chained together with

the first element in each row indicating the next node or the last node of a partition. The last two elements of the row are used for the final color assigned and the partition color assigned, respectively. A second matrix contains the first node of the partitions and the number of colors in the partition solutions. Thus, the two matrices store the information about the graph and its partitions, for all subroutines.

1. The Look-ahead Algorithm

To evaluate the effectiveness of the Look-ahead algorithm, the sequence of graphs in Table I were generated. Graphs 1 thru 10 were generated without partitioning. Graphs 11 thru 20 were generated with partitioning, but were colored as a whole graph. The graphs were colored, using the Look-ahead algorithm, with the nodes ordered in descending order by degree. In addition, these graphs were colored, using the Welsh-Powell algorithm as well as the Look-ahead algorithm, with the nodes in the generated order. The results are listed in Table I.

The graphs were colored, using the Look-ahead algorithm with nodes in the order as generated, to observe the affect of ordering the nodes by degree, on the Look-ahead algorithm. Ordering the nodes by degree causes the Look-ahead algorithm to assign colors to the nodes with the highest degree first. Thus, a maximum number of nodes in the graph are connected to nodes with previously assigned colors. Due to this method of assigning colors, forward

Graph No.	Order	Percent of Connections Int.	Ext.	No. of Partitions	Number of Colors Look-ahead Unordered	Ordered	Welsh-Powell
1	80	.7	0	1	26	23	24
2	80	.6	0	1	19	20	21
3	80	.5	0	1	16	16	17
4	80	.4	0	1	14	14	15
5	70	.7	0	1	23	22	21
6	70	.6	0	1	20	19	19
7	70	.5	0	1	16	15	15
8	70	.4	0	1	13	11	11
9	60	.5	0	1	13	14	12
10	60	.4	0	1	13	12	12
11	80	.4	.2	5	10	9	10
12	80	.5	.1	4	9	9	9
13	70	.4	.2	4	10	9	8
14	70	.5	.1	4	8	8	8
15	60	.4	.1	3	9	7	8
16	60	.6	.2	4	11	10	10
17	60	.5	.1	4	8	8	8
18	50	.7	.2	4	10	8	9
19	50	.6	.1	2	10	10	9
20	50	.6	.1	2	10	10	9

TABLE I. A Comparison of the Look-ahead Algorithm with the Welsh-Powell Algorithm.

The parameter values used in generating the partitioned graphs were obtained by processing the second quarter 1967-1968 schedule for the Naval Postgraduate School. The course conflicts were tabulated by department code. This tabulation was then used to group courses, by department code, to achieve a partitioned conflict graph with relatively high internal connectivity and relatively low external connectivity. The results are contained in Appendix A.

The parameters of the random graph generator were varied individually to observe the effect on the solutions, as obtained by the two algorithms. Specifically, the effect of the ratio, external connectivity to internal connectivity, and the number of partitions in the graph, were observed. In addition, the same graphs were colored as complete graphs, using the Welsh-Powell and Look-ahead algorithms, to compare the solutions with the solutions achieved by combining partitions. The results are contained in Table II.

No.	ord.	Percent of conn. Int.	Ext.	No. of part's	Look-ahead Ord.	Unord.	Welsh-Powell	Combination Match	Graph color
1	100	.5	.1	2	12	14	14	15	15
2	100	.5	.1	3	11	12	11	14	13
3	100	.5	.1	4	10	11	12	14	12
4	100	.5	.1	5	9	10	10	13	13
5	100	.5	.1	6	8	10	10	11	11
6	100	.5	.1	7	9	10	10	11	11
7	100	.1	.1	4	6	6	6	9	9
8	100	.2	.1	4	7	7	8	13	13
9	100	.3	.1	4	8	9	8	14	13
10	100	.4	.1	4	9	10	10	12	11
11	100	.5	.1	4	10	11	12	14	12
12	100	.6	.1	4	12	14	12	12	12
13	50	.5	.1	3	7	8	7	8	7
14	50	.6	.1	2	10	10	9	9	9
15	50	.7	.2	4	8	10	9	10	9
16	60	.5	.1	4	8	8	8	9	8
17	60	.5	.2	4	10	11	10	12	11
18	60	.4	.1	3	7	9	8	10	9
19	70	.5	.1	4	8	8	8	9	8
20	70	.4	.2	4	9	10	8	12	12
21	80	.5	.1	4	9	9	9	11	11
22	80	.4	.2	5	9	10	10	15	15

TABLE II. Comparison of Partitioned Coloring with Non-partitioned Coloring.

IV. CONCLUSIONS

To be of value in the class scheduling applications, the graph coloring method should give solutions which are no worse than solutions obtained by simple sequential coloring methods. Since the desired results were not achieved on modernate order graphs, as indicated in Table II, the methods developed are not considered appropriate to class scheduling applications.

The two methods developed for coloring graphs by partitions failed to produce desired results due to the inability to select alternate solutions. The coloring solutions within the partitions were considered as being fixed. Thus, possible alternate solutions, within the partitions, were not considered in the construction of the coloring solution for the entire graph. In addition, alternate combinations of the partition solutions were not considered by the two methods. The failure of the two methods was caused by this non-selectivity in attaining a solution for the entire graph.

The fixed partition solutions generate a large number of edges in the coalesced graphs. Thus, in the matching algorithm there is an insufficient number of edges in the bipartite graph. This results in too many colors of the partitions being combined, requiring new final colors. In the coalesced graph coloring algorithm, the large number of edges generated requires a large number of colors, when the

coalesced graph is colored. In both algorithms, the end result contains too many colors.

A number of the original graphs and associated coalesced graphs were analyzed. This revealed that the connectivity of the coalesced graphs could be reduced by selecting alternate solutions for the partitions, thus producing acceptable results. In some cases, the selection of an alternate solution at the time when the partitions were combined produced acceptable results. In all cases analyzed, a more complex algorithm could produce acceptable results. This algorithm would need the capability to select alternate solutions, in both the partitions and at the stage of the algorithm when the partition solutions are combined.

Although the two methods of coloring graphs by partitions failed to produce acceptable results, the Look-ahead coloring algorithm produced acceptable results in coloring graphs. A comparison of the coloring for the graphs, listed in Table I and Table II, shows the Look-ahead method gives a coloring with less than or equal to the number of colors in the Welsh-Powell solution for 28 of the 32 different graphs. The Look-ahead solution contains less colors than the Welsh-Powell solution for 15 of the 32 graphs. In the tests on graphs of order 20 and 50, where a large number of graphs were colored, the Look-ahead algorithm's solution was equivalent to or better than the Welsh-Powell solution for 97.7 percent of the graphs of order 20. The Look-ahead solution gave fewer colors in 14.7 percent of the graphs.

For graphs of order 50, the Look-ahead algorithm produced an equivalent or better solution for 88.6 percent of the graphs, with 38.6 percent of the solutions containing fewer colors than the Welsh-Powell solution.

Wood [7] describes a method of coloring graphs, using a similarity matrix. This method colors the graph by pairing nodes of greatest similarity. The similarity matrix solutions were compared with the Welsh-Powell solutions for 100 random graphs of order 20, 50, and 100, and connectivities of .25, .50, and .75. The results, as listed in Reference 7, for the graphs of order 20 shows the similarity matrix solutions are equivalent or better for 82 percent of the graphs, with 21 percent better than the Welsh-Powell solutions. For the graphs of order 50, the similarity matrix solutions were equivalent or better for 78 percent of the graphs, with 28 percent better than Welsh-Powell solutions. The significant factor, for the graphs of order 50, is that the Welsh-Powell algorithm gives better results at low connectivity, .25, whereas the similarity matrix method gives better results at high connectivity, .75. Wood claims that his similarity matrix method is an improvement over the Welsh-Powell algorithm, except for large order low connectivity graphs.

A comparison of the data for the Look-ahead algorithm with the data for the similarity matrix method, in Table III, shows the following:

(1) The solutions achieved by the similarity matrix method, are the best solutions for a slightly higher

percentage of graphs of order 20 than the Look-ahead method.

(2) The Welsh-Powell algorithm achieves the best solution for a significantly higher percentage of the graphs in the comparison with the similarity matrix method, than in the comparison with the Look-ahead algorithm.

(3) The Look-ahead algorithm achieves a significantly higher percentage of solutions in the category of equivalent or better, than the similarity matrix method. In addition, the Look-ahead algorithm does not yield to the Welsh-Powell algorithm, for any order or connectivity tested. Thus, the Look-ahead algorithm is considered an improvement over both the Welsh-Powell algorithm and the similarity matrix method.

Although the Look-ahead algorithm was designed for coloring partitions, it may be possible to achieve class schedules using the Look-ahead algorithm to color the complete conflict graph. A graph was constructed using the courses of the third group, listed in Appendix A, of the Naval Postgraduate School schedule for the second quarter of 1967-1968 academic year. These courses were considered as an hourly conflict graph. The associated conflict graph was colored using the Look-ahead algorithm and sequential colors. The graph was colored with 20 colors. Thus, with the implementation of color vectors in the Look-ahead algorithm, it may be possible to achieve acceptable class schedules.

Order	Percent of conn.	Comparison of similarity matrix/Look-ahead methods with the Welsh-Powell method. Data listed is percent of solutions.					
		Better		Same		Worse	
		Sim	LA	Sim	LA	Sim	LA
20	.25	21	16	61	83	18	1
20	.50	26	16	54	79	20	5
20	.75	21	12	67	87	12	1
50	.25	8	36	66	48	26	6
50	.50	30	36	52	38	18	26
50	.75	40	44	31	54	21	2

TABLE III. Comparison of Three Methods.

APPENDIX A: SCHEDULE DATA

The Naval Postgraduate School schedule for the second quarter 1967-1968 academic year was processed to obtain the following data: The schedule contained 424 courses with a total of 1373 class hours. This is an average of 3.24 hours per course. The course conflicts were tabulated by department code. This tabulation was used to group courses offered by departments, to achieve the following statistics:

Group	Number of Courses	Edges existing Int.	Edges existing Ext.	Maximum edges possible Int.	Maximum edges possible Ext.	Percent of edges Int.	Percent of edges Ext.
AO, BI, AE	48	126	51	2256	19336	.056	.003
CS, LT, MN, SP	43	89	168	1806	16555	.049	.010
CO, CH, GV, HI, ME, MR, MS, NW	115	227	335	13110	36225	.017	.009
EE, EN, MA, OA, PH, PS	224	640	392	49952	46144	.015	.008

Average percent of internal edges..... .034

Average percent of external edges..... .008

Average ratio of internal edges to external edges..... 4.25:1

APPENDIX B

PL/1 LISTINGS FOR ALGORITHMS

```
COLOR: PROC (LISTIN);
  /*                LOOKAHEAD ALGORITHM
    THE GRAPH IS CONTAINED IN A COMMON DATA AREA NAMED
CONGRAPH, DECLARED IN THE MAIN PROGRAM.  KOL IS A COMMON
DATA AREA USED TO PASS THE NUMBER OF COLORS TO THE MAIN
PROGRAM*/
DCL (LISTIN,/* FIRST NODE IN LIST*/
    NODE,/* CURRENT NODE*/
    TNODE,/*CURRENT CONFLICT NODE*/
    SNOD ,/* CURRENT SECOND LEVEL NODE*/
    NAVL,/*NUMBER OF COLORS AVAILIABLE FOR ASSIGNMENT*/
    MAXCOL,/*NUMBER OF COLORS PRESENTLY USED*/
    CCL,/*COLOR OF SECOND LEVEL CONFLICT NODE*/
    MAXSCON,/*COLOR WITH MAXIMUM SECOND LEVEL CONFLICTS*/
    SCONS,/*MAXIMUM SECOND LEVEL CONFLICTS*/
    NXC,/*NEXT CONFLICT LOCATION*/
    NOCO,/*NUMBER OF NODES WITH NO COLOR ASSIGNED*/
    COLIST(45),/*LIST OF COLORS*/
    NOCOLIST(100),/*LIST OF NODES WITH NO COLOR ASSIGNMENT*/
    AVCOL(45),/*LIST OF COLORS AVAILIABLE FOR ASSIGNMENT*/
    I /*LOOP VARIABLE*/
    ) BIN FIXED(15);
    NODE =LISTIN;
    DO WHILE(NODE ¬= 9999); /*ZERO COLOR CELLS */
        CONGRAPH(NODE,97)=0;
        CONGRAPH(NODE,98)=0;
        NODE=CONGRAPH(NODE,0);
        END;
    NODE=LISTIN;
    CONGRAPH(LISTIN,98),MAXCOL=1; /*ASSIGN FIRST COLOR*/
    DO WHILE (CONGRAPH(NODE,0) ¬= 9999);
  /*GET NEXT NODE*/
        NODE=CONGRAPH(NODE,0);
        DO I=1 TO MAXCOL; /* RESET COLOR FLAGS*/
            COLIST(I) = 0;
            END;
        NOCO=0; /* RESET NOCOLOR LIST*/
  /*BUILD LIST OF CONFLICT COLORS AND CONFLICT NODES WITH */
  /*NO COLOR*/
        DO NXC =1 TO 95 WHILE(CONGRAPH(NODE,NXC) ¬= 0);
  /*CONFLICT NODE*/
            TNODE = CONGRAPH(NODE,NXC);
  /*IS NODE IN CURRENT SECTION*/
            IF CONGRAPH(TNODE,0) > 0 THEN DO;
  /*HAS NODE BEEN ASSIGNED A COLOR*/
                IF CONGRAPH(TNODE,98)>0 THEN
  /*MARK COLOR NOT AVAILIABLE*/
                COLIST(CONGRAPH(TNODE,98))=1;
                ELSE DO;
  /*PLACE ON THE NO COLOR LIST*/
                    NOCO = NOCO+1;
                    NOCOLIST(NOCO)=TNODE;
                    END;
                END;
            END;
        NAVL=0; /* SET NO OF AVAILIABLE COLORS TO ZERO*/
        DO I=1 TO MAXCOL; /* GET AVAILIABLE COLORS*/
            IF COLIST(I) =0 THEN DO; /* COLOR AVAILIABLE*/
                NAVL = NAVL + 1;/*COUNT NO OF COLORS */
                AVCOL(NAVL)=I; /* BUILD LIST OF AVAILIABLE
                                  COLORS*/
                END;
            END;
```

```
/*   IF THE NUMBER OF COLORS AVAILIABLE FOR ASSIGNMENT IS;
  1. ONE, ASSIGN COLOR TO NODE.
  2. GREATER THAN ONE, ASSIGN COLOR WITH MAXIMUM SECOND
  LEVEL CONFLICTS.
  3. ZERO, INCREASE THE CURRENT NUMBER OF COLORS AND
  ASSIGN THE NEW COLOR.  */
      IF NAVL > 0 THEN DO;  /*COLORS AVAIL*/
         IF NAVL>1 THEN DO;/*MULTI-COLORS AVAILIABLE*/
/*CHECK SECOND LEVEL CONFLICTS*/
            DO N=1 TO NOCO;
               SNOD=NOCOLIST(N); /* SECOND LEVEL NODE*/
               DO I=1 TO 95 WHILE(CONGRAPH(SNOD,I)¬=0);
/*SECOND LEVEL CONFLICT NODE*/
               TNODE=CONGRAPH(SNOD,I);
/*IS THE NODE IN CURRENT PARTITION*/
               IF CONGRAPH(TNODE,0)>0 THEN
/*HAS A COLOR BEEN ASSIGNED*/
               IF CONGRAPH(TNODE,98) > 0 THEN DO;
/*FORWARD CONFLICT COLOR*/
                  COL=CONGRAPH(TNODE,98);
                  IF COLIST(COL) <= 0 THEN
/*ADD TO FORWARD COLOR CONFLICT LIST*/
                    COLIST(COL)=COLIST(COL)-1;
                  END;
               END;
            END;
/*GET COLOR CONTRIBUTED BY THE MAXIMUM NUMBER OF NODES*/
            SCONS =COLIST(AVCOL(1));
            MAXSCON = AVCOL(1);
            DO I=2 TO NAVL;
               IF COLIST(AVCOL(I)) < SCONS THEN DO;
                  MAXSCON= AVCOL(I);
                  SCONS = COLIST(MAXSCON);
                  END;
               END;
/*ASSIGN COLOR WITH MAXIMUM NUMBER OF FORWARD CONFLICTS*/
            CONGRAPH(NODE,98)=MAXSCON;
            END;
/*ONLY ONE COLOR AVAILIABLE*/
         ELSE CONGRAPH(NODE,98)=AVCOL(1);
         END;
      ELSE DO; /* NO AVAILIABLE COLORS*/
         MAXCOL = MAXCOL + 1;
         CONGRAPH(NODE,98) = MAXCOL;
         END;
      END;
   KOL=MAXCOL;/*SET NUMBER OF COLORS FOR MAIN PROGRAM*/
   END COLOR;
```

```
MATCH: PROC(P1,P2);
  /*          MATCHING ALGORITHM
    THIS ROUTINE SUCCESSIVELY MATCHES THE PARTITION SOLUTION
COLORS TO THE FINAL COLORS ASSIGNED TO THE ALREADY COMBINED
PARTITIONS.  A BIPARTITE GRAPH IS BUILT (BIGRP) USING THE
DATA FROM THE ORIGINAL GRAPH.  BIGRP IA A SQUARE MATRIX WITH
ROW I AND COLUMN I REPRESENTING THE SAME COLOR.  THE FINAL
COLORS OF THE COMBINED PARTITIONS ARE REPRESENTED BY NUMBERS
1 THRU 50 AND THE PARTITION COLORS OF THE NEXT PARTITION ARE
REPRESENTED BY NUMBERS 51 THRU LIMIT OF GRAPH.  THUS, IF
PARTITION COLOR 1 IS CONNECTED TO A NODE WITH FINAL COLOR 7
THEN BIGRP(7,51)=BIGRP(51,7)=1, AND THE EDGE CONNECTIONS, IN
THE BIPARTITE GRAPH, ARE THE ZERO ELEMENTS OF MATRIX BIGRP.
THE ORIGINAL GRAPH, PARTITION STARTING NODES, AND THE NUMBER
OF COLORS FOR THE PARTITIONS ARE IN COMMON DATA AREAS, CON-
GRAPH, PART(I,1), AND PART(I,2), RESPECTIVELY.  THE NUMBER
OF FINAL COLORS ASSIGNED  COMBINED PARTITIONS IS COMMON
DATA, KOL.*/
  DCL ALCHN ENTRY(BIN FIXED(15));/*A RECURSIVE ROUTINE FOR
    FINDING THE ALTERNATING CHAINS IN THE BIPARTITE GRAPH*/
  DCL (BIGRP(1:U,1:100),/*THE BIPARTITE GRAPH*/
    COMCOL,/*LOCAL NUMBER OF FINAL COLORS*/
    BILIM,/*UPPER LIMIT OF THE BIPARTITE GRAPH*/
    ALTC(100),/* LIST OF NODES IN THE ALTERNATING CHAIN*/
    I,J,K,L,M,N,/*LOOP AND TEMPORARY VARIABLES*/
    P1,/*ENTRY PARAMETER, POINTS TO THE FIRST NODE OF THE
    COMBINED PARTITIONS*/
    P2,/*ENTRY PARAMETER, POINTS TO FIRST NODE OF THE NEXT
    PARTITION*/
    ND,/*CURRENT NODE*/
    CND,/*CURRENT CONFLICT NODE*/
    NDCL,/*CURRENT NODE COLOR*/
    CNDCR,/*CURRENT CONFLICT NODE COLOR*/
    CMP/*PARTITION COLOR INDICATOR*/
    )BIN FIXED(15);
  DCL(FAC,/*FLAG USED TO INDICATE PRESENCE OF AN ALTERNATING
    CHAIN*/
    IN /*FLAG USED TO INDICATE IF THE NEXT EDGE SHOULD BE IN
    THE PRESENT SOLUTION*/
    )BIT(1);
  /*NUMBER OF COLORS IN COMBINED PARTITIONS*/
    COMCOL=KOL;
  /*SET UPPER LIMIT OF GRAPH*/
    BILIM=PART(P1,2)+50;
  /*INITIALIZE MATRIX ELEMENTS TO 1 EXCEPT WHERE AN EDGE IN
  THE BIPARTITE GRAPH IS POSSIBLE,  INITIALIZE THOSE TO 0*/
    DO I=1 TO BILIM;
         IF I<=COMCOL | I>50 THEN BIGRP(I,0)=0;
         DO J=1 TO BILIM;
              IF (J>50&I<=COMCOL)|(J<=COMCOL&I>50) THEN
              BIGRP(I,J)=0;
              ELSE BIGRP(I,J)=1;
              END;
         END;
  /*COMPUTE PARTITION COLOR INDICATOR,CMP=(100*PARTITION
  NUMBER)*/
    CMP = 0;
    NDCL=CONGRAPH(P2,98);
    DO WHILE (NDCL>100);
         NDCL=NDCL-100;
         CMP=CMP+100;
         END;
    ND=P2;
  /* BUILD THE BIPARTITE GRAPH*/
    DO WHILE (ND¬=9999);
  /* THE NODE'S PARTITION COLOR*/
         NDCL=CONGRAPH(ND,98)+50-CMP;
         DO CND=1 TO 100 WHILE(CONGRAPH(ND,CND)¬=0);
  /*CONFLICT NODE'S FINAL COLOR*/
              CNDCR=CONGRAPH(CONGRAPH(ND,CND),97);
```

```
  /*HAS A FINAL COLOR BEEN  ASSIGNED*/
              IF CNDCR>0 THEN DO;
                  BIGRP(NDCL,CNDCR)=1;
                  BIGRP(CNDCR,NDCL)=1;
                  END;
              END;
  /* NEXT NODE IN THE PARTITION*/
         ND=-CONGRAPH(ND,0);
         END;
    FAC='1'B;
  /* FIND ALTERNATING CHAINS IN THE BIPARTITE GRAPH*/
    DO WHILE(FAC);
  /*RESET ALTERNATING CHAIN FLAG*/
         FAC='0'B;
  /*FIND STARTING NODE IN ALTERNATING CHAIN*/
         DO I=1 TO BILIM WHILE (¬FAC);
              IF BIGRP(I,0)=0 THEN DO;
                  K=1;
  /*ASSIGN THE FIRST NODE IN ALTERNATING CHAIN*/
                  ALTC(K)=I;
                  BIGRP(I,0)=-1;
  /*THE NEXT EDGE MUST BE IN PRESENT SOLUTION*/
                  IN='1'B;
  /*CALL ROUTINE TO COMPLETE THE ALTERNATING CHAIN*/
                  CALL ALCHN(I);
                  BIGRP(I,0)=0;
                  END;
              END;
  /* IF AN ALTERNATING CHAIN IS FOUND, UPDATE PRESENT
  SOLUTION*/
         IF FAC THEN DO;
              DO I=1 TO LIM WHILE(ALTC(I+1)¬=0);
  /*GET END POINTS OF THE EDGE (L&M)*/
                  L =ALTC(I);
                  M=ALTC(I+1);
  /* IF I IS ODD ADD THE EDGE TO PRESENT SOLUTION*/
                  IF MOD(I,2)>0 THEN DO;
                       BIGRP(L,2)=M;
                       BIGRP(M,2)=L;
                       BIGRP(L,M)=-1;
                       BIGRP(M,L)=-1;
                       END;
                  ELSE DO;
  /* IF I IS EVEN REMOVE THE EDGE FROM PRESENT SOLUTION*/
                       BIGRP(L,M)=0;
                       BIGRP(M,L)=0;
                       END;
                  END;
              END;
  /* ADD COLORS TO COMBINED PARTITION TO ALLOW FOR THE UN-
  MATCHED COLORS OF THE PARTITION BEING ADDED*/
    DO I=51 TO BILIM;
         IF BIGRP(I,0)=0 THEN DO;
              KOL=KOL+1;
              BIGRP(I,0)=KOL;
              END;
         END;
    ND=P2;
  /* ASSIGN FINAL COLORS TO PARTITION JUST COMBINED*/
    DO WHILE(ND¬=9999);
         NDCL=CONGRAPH(ND,98)+50-CMP;
         CONGRAPH(ND,97)=BIGRP(NDCL,0);
         ND=-CONGRAPH(ND,0);
         END;
ALCHN: PROC(ND) RECURSIVE;
  /* ROUTINE FOR FINDING ALTERNATING CHAINS*/
  /*VARIABLES USED IN RECURSION*/
DCL (J,ND) BIN FIXED(15);
    DO J=1 TO BILIM WHILE(¬FAC);
  /*FIND EDGE NOT IN THE PRESENT SOLUTION*/
         IF BIGRP(ND,J)=0 & IN THEN DO;
```

```
/*DOES THE EDGE TERMINATE ON AN UN-MATCHED NODE*/
          IF BIGRP(J,0)=0 THEN DO;
/* ADD THE NODE TO THE LIST*/
               K=K+1;
               ALTC(K)=J;
/* MARK END OF LIST*/
               ALTC(K+1)=0;
/* SET FOUND ALTERNATING CHAIN FLAG*/
               FAC='1'B;
               RETURN;
               END;
/* IS NODE ALREADY IN THE LIST*/
          ELSE IF BIGRP(J,1)>0 THEN DO;
/*SET FLAG TO LOOK FOR AN EDGE TO REMOVE FROM PRESENT
SOLUTION*/
               IN=¬IN;
/* ADD NODE TO LIST*/
               K=K+1;
               ALTC(K)=J;
               BIGRP(J,1)=-BIGRP(J,1);
/* CONTINUE THE SEARCH FOR AN ALTERNATING CHAIN*/
               CALL ALCHN(J);
               BIGRP(J,1)=-BIGRP(J,1);
               END;
          END;
/*CAN EDGE BE REMOVED FROM PRESENT SOLUTION*/
     IF BIGRP(ND,J)=-1 & ¬IN THEN DO;
          K=K+1;
/*SET FLAG TO LOOK FOR AN EDGE TO ADD TO PRESENT SOLUTION
*/
          IN=¬IN;
/*ADD NODE TO LIST*/
          ALTC(K)=J;
          BIGRP(J,1)=-BIGRP(J,1);
/* CONTINUE THE SEARCH FOR AN ALTERNATING CHAIN*/
          CALL ALCHN(J);
          BIGRP(J,1)=-BIGRP(J,1);
          END;
     END;
/* HAS AN ALTERNATING CHAIN BEEN FOUND*/
  IF FAC THEN RETURN;
/* DELETE LAST NODE FROM LIST*/
  K=K-1;
/* RESET FLAG FOR PREVIOUS NODE IN LIST TO CONTINUE THE
SEARCH FOR AN ALTERNATING CHAIN*/
  IN=¬IN;
  RETURN;
  END ALCHN;
  END MATCH;
```

```
PARCCL: PROC;
  /*          COALESCED GRAPH COLORING ALGORITHM
    THIS ROUTINE COMBINES PARTITION SOLUTION BY CONSTRUCTING
  A COALESCED GRAPH USING PARTITION COLORS AND THE EXTERNAL
  CONNECTIONS OF THE PARTITIONS.  THE COALESCED GRAPH IS
  CONSTRUCTED IN THE COMMON DATA AREA, CONGRAPH, ABOVE THE
  LAST NODE OF THE ORIGINAL GRAPH.  THE FIRST NODE OF THE
  PARTITIONS AND THE NUMBER OF COLORS IN THE PARTITION
  SOLUTIONS ARE CONTAINED IN COMMON DATA PART(I,1) AND
  PART(I,2), RESPECTIVELY.  ORD IS COMMON DATA AND IS THE
  ORDER OF THE ORIGINAL GRAPH*/
  DCL(KST(10),/*THE FIRST NODE OF THE PARTITION COLORS IN
    THE COALESCED GRAPH*/
    ORL(100),/*TEMPORARY STORAGE FOR CONFLICT NODES*/
    ND,/*CURRENT NODE*/
    CND,/*CONFLICT NODE*/
    NDK,/*CURRENT NODE COLOR*/
    CNDK,/*CONFLICT NODE COLOR*/
    UL,/*UPPER LIMIT OF COALESCED GRAPH*/
  I,J,K,L,M,N,SC1,SC2/*LOOP AND TEMPORARY VARIABLES*/
    ) BIN FIXED(15);
  /* SET PARTITION COLOR STARTING NODES*/
    KST(1)=ORD;
    DO I=2 TO NG;
        KST(I)=KST(I-1)+PART(I-1,2) ;
        END;
  /*SET UPPER LIMIT OF COALESCED GRAPH*/
    UL =KST(NG)+PART(NG,2);
    KST(NG+1)=UL;
    /* CLEAR CONFLICT COUNT    */
    DO I=ORD+1 TO UL;
        CONGRAPH(I,98)=0;
  /* CHAIN THE NODES OF THE COALESCED GRAPH*/
        CONGRAPH(I,0)=I+1;
        END;
  /*CONSTRUCT THE COALESCED GRAPH*/
    DO ND=1 TO ORD;
  /*PARTITION COLOR OF NODE*/
        NDK=CONGRAPH(ND,98);
  /*EXTRACT THE PARTITION NUMBER FOR THE NODE*/
        SC1=NDK/100;
        DO CND=1 TO 95 WHILE(CONGRAPH(ND,CND)¬=0);
  /*PARTITION COLOR OF CONFLICT NODE*/
            CNDK=CONGRAPH(CONGRAPH(ND,CND),98);
  /*EXTRACT PARTITION NUMBER FOR THE CONFLICT NODE*/
            SC2=CNDK/100;
  /*ARE THE NODES IN THE SAME PARTITION*/
            IF SC1¬=SC2 THEN DO;
  /*COMPUTE THE NODES IN THE COALESCED GRAPH, FOR THE TWO
  PARTITION COLORS*/
                M=KST(SC1)+MOD(NDK,100);
                N=KST(SC2)+MOD(CNDK,100);
                J=0;
  /*IS THE CONFLICT IN THE COALESCED GRAPH*/
                DO I=1 TO CONGRAPH(M,98) WHILE(J=0);
                    IF CONGRAPH(M,I)=N THEN J=I;
                    END;
  /*ADD THE CONFLICT TO THE COALESCED GRAPH AND INDEX THE
  CONFLICT COUNT ON THE NODE*/
                IF J=0 THEN DO;
                    J=CONGRAPH(M,98)+1;
                    CONGRAPH(M,98)=J;
                    CONGRAPH(M,J)=N;
                    J=CONGRAPH(N,98)+1;
                    CONGRAPH(N,98)=J;
                    CONGRAPH(N,J)=M;
                    END;
                END;
            END;
        END;
```

```
/*BUILD COMPLETE SUBGRAPH ON THE PARTITION COLOR SOLUTIONS
*/
/*SET THE PARTITION NUMBER*/
  N=1;
  DO ND=ORD+1 TO UL;
/*PLACE INTER-PARTITION CONFLICTS IN THE ORDERING LIST*/
      DO CND=1 TO CONGRAPH(ND,98);
           ORL(CND)=CONGRAPH(ND,CND);
           END;
/*ADD CONFLICTS FOR THE COMPLETE SUB-GRAPH ON PARTITION
COLORS*/
      DO J=KST(N)+1 TO ND-1,ND+1 TO KST(N+1);
           ORL(CND)=J;
           CND=CND+1;
           END;
/*ORDER LIST OF CONFLICTS IN ASCENDING ORDER*/
      J=1;
      DO WHILE(J>0);
           J=0;
           DO I=1 TO CND-2;
                IF ORL(I)>ORL(I+1) THEN DO;
                     M=ORL(I);
                     ORL(I)=ORL(I+1);
                     ORL(I+1)=M;
                     J=1;
                     END;
                END;
           END;
/*PLACE ORDERED LIST IN THE COALESCED GRAPH*/
      DO J=1 TO CND-1;
           CONGRAPH(ND,J)=ORL(J);
           END;
/*MARK THE END OF THE CONFLICT LIST*/
      CONGRAPH(ND,CND)=0;
/* IF THE NODE IS THE LAST NODE FOR A PARTITION, INDEX
THE PARTITION NUMBER*/
      IF ND=KST(N+1) THEN N=N+1;
      END;
/*MARK THE LAST NODE OF THE COALESCED GRAPH*/
  CONGRAPH(UL,0)=9999;
/*SET PARAMETER FOR COLOR ROUTINE*/
  PT=NG+1;
/*ORDER THE NODES OF THE COALESCED GRAPH IN DESCENDING
ORDER BY DEGREE*/
  CALL ORDER(ORD+1);
/*COLOR THE COALESCED GRAPH*/
  CALL COLOR(PART(PT,1));
/* ASSIGN FINAL COLORS TO ORIGINAL GRAPH*/
  DO I=1 TO ORD;
/*PARTITION COLOR OF NODE*/
      NDK=CONGRAPH(I,98);
/* COMPUTE THE REPRESENTATIVE NODE IN THE COALESCED GRAPH*
AND GET FINAL COLOR*/
      SC1=NDK/100;
      SC2=MOD(NDK,100);
      M=CONGRAPH(KST(SC1)+SC2,98);
/* ASSIGN THE FINAL COLOR*/
      CONGRAPH(I,97)=M;
      END;
  END PARCOL;
```

LIST OF REFERENCES

1. Appleby, J. S., Blake, D. V. and Newman, E. A., "Techniques for Producing School Timetables on a Computer and Their Applications to Other Scheduling Problems,: Computer Journal, v. 3, p. 237-245, January 1961.

2. Busacker, R. G., and Saaty, T. L., Finite Graphs and Networks: An Introduction with Applications, McGraw-Hill Book Company, 1965.

3. Csima, J., and Gottlieb, C. C., "Tests on a Computer Method for Constructing School Timetables," Communications of the Association for Computing Machinery, v. 3, p. 160-163, March 1964.

4. Mack, J. A., III, An Application of Graph Coloring to a Scheduling Problem, M. S. Thesis, Naval Postgraduate School, Monterey, California, June 1968.

5. Ore, O., Theory of Graphs, Colloquium Publications, American Mathematical Society, 1962.

6. Welsh, D. J. A., and Powell, M. B., "An Upper Bound for the Chromatic Number of a Graph and its Applicability to Timetabling Problems," Computer Journal, v. 10, p. 85-86, May 1967.

7. Wood, D. C., "A Technique for Coloring a Graph to Large Timetabling Problems," Computer Journal, v. 12, p. 317-319, November 1969.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Assoc Professor U. R. Kodres, Code 53Kr Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4.	LCDR E. A. Singer, Code 53Sf Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
5.	LT Charles L. Deitrick, USN R.D. #2 Milton, Pennsylvania 17847	1
6.	Dr. B. J. Lockhart, Code 022 Dean of Curricula Naval Postgraduate School Monterey, California 93940	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author)

Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION

UNCLASSIFIED

2b. GROUP

3 REPORT TITLE

A GRAPH THEORETIC APPROACH TO THE CLASS SCHEDULING PROBLEM

4 DESCRIPTIVE NOTES (Type of report and inclusive dates)

Master's Thesis; June 1971

5. AUTHOR(S) (First name, middle initial, last name)

Charles L. Deitrick

6. REPORT DATE

June 1971

7a. TOTAL NO. OF PAGES

53

7b. NO. OF REFS

7

8a. CONTRACT OR GRANT NO.

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10 DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY

Naval Postgraduate School
Monterey, California 93940

13. ABSTRACT

Two algorithms for coloring large order graphs by partitioning, as related to class scheduling with a computer, are developed. Although, the two main algorithms failed to produce acceptable results for application to class scheduling, a coloring algorithm developed for use in the two main algorithms, is an improvement over known existing coloring algorithms.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

A-31408

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Graph Theory						
Graph Coloring						
Class Scheduling						
Timetable						

Thesis
D255 Deitrick
c.1 A graph theoretic approach to the class scheduling problem.